Framework de aplicaciones para red social colaborativa distribuida

Pablo Martínez Bernardo

Grado en Ingeniería Informática
Facultad de Informática

UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo de Fin de Grado
Madrid, junio 2014

Directores:
Samer Hassan Collado
Antonio Tenorio Fornés

Authorization of dissemination and usage

Autorizo a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor, tanto la propia memoria, como el código, la documentación y/o el software desarrollado.

Pablo Martínez Bernardo

Madrid, junio 2014

Contents

List of Figures

List of Tables

Abstract

Real-time online collaborative software grants the user a broad collection of alternatives including, yet not limited to, concurrent document edition, gaming and technical applications. It is a field of increasing importance and still not fully exploited potential.

Apache Wave provides a standard for federated real-time collaboration allowing the usage of small applications with collaborative purpose named Gadgets, embedded in particular online documents. However the amount of activity regarding Apache Wave declined in the recent past and there exists a concerning lack of information relative to the development of Gadgets. Accompanying that, there are also important needs solvable with Gadgets that have not been implemented.

This work explores the scene of Wave, and specifically attemps to set the base for real-world developers to code collaborative applications under the Wave infrastructure. In the frame of this work Several different extensions to Apache Wave have been developed taking advantage of open source software, open protocols and open standards. Besides, undocumented features and procedures to help developers have been detailed.

The developed extensions are the following: a gadget to set an open license to your content, a gadget for decision making, a gadget for video conference, and another extension for highlighting the contributions of each participant in a document. Information about the development and use of extensions has been documented and shared publicly.

The results of this project constitute a framework that aids developers attemping to code extensions for Apache Wave by gathering and creating information about the process, as well as implementing four extensions that are both needed in the community and representative of key functionalities.

Keywords:
Apache Wave, Collaboration, Gadgets, Apps, Real Time, Federation, Google Web Toolkit

Resumen

El software colaborativo en tiempo real le otorga al usuario una amplia variedad de alternativas, entre ellas edición concurrente de documentos, vídeo juegos y aplicaciones técnicas. Es un campo de creciente importancia y todavía no explotado completamente.

Apache Wave facilita un estándar para colaboración federada en tiempo real permitiendo el uso de pequeñas aplicaciones colaborativas llamadas Gadgets, integradas en ciertos documentos online. La actividad en cuanto a Apache Wave ha ido disminuyendo recientemente y existe una preocupante falta de información relativa al desarrollo de Gadgets. Asimismo, también hay importantes necesidades solventables con Gadgets que todavía no han sido resueltas.

Este trabajo explora el panorama de Wave, y concretamente intenta establecer la base para que los desarolladores del mundo real puedan programar aplicaciones colaborativas bajo la infraestructura Wave. En el marco de este trabajo diversas extensiones de Apache Wave han sido desarolladas haciendo uso de software libre, así como protocolos y estándares libres. Asimismo, características previamente no documentadas y procedimientos para ayudar a los desarrolladores han sido detallados.

Las extensiones desarrolladas son las siguientes: un Gadget para asignar una licencia libre a tu propio contenido, un Gadget de toma de decisiones, un Gadget para videoconferencia, y otra extensión para colorear las contribuciones de cada participante en un documento. Se ha documentado y compartido públicamente información sobre el desarrollo y uso de las extensiones.

El resultado de este proyecto genera un framework el cual facilita a los desarrolladores pretendiendo programar extensiones de Apache Wave recolectando y creando información acerca del proceso, además de implementar extensiones necesarias en la comunidad y representativas de funcionalidades clave.

Palabras Clave:
Apache Wave, Colaboración, Gadgets, Apps, Tiempo Real, Federación, Google Web Toolkit

Chapter 1

Introduction

Nowadays online social networks such as Facebook and Google+ are a big part of the mainstream Internet usage. There have been countless social networks in a very short span of time, and each time they are different and innovative, and are growing importance offline.

One of those remarkable attempts was **Google Wave**, that had a short life before Google dropped support and development, and even took the working nodes down. However the control was given to Apache to continue development of it. Since the inception of Wave, the web has undergone several changes, but the technology and protocol still remain relevant. JavaScript is one of the most used resources to execute code on the client-side

. The world has also faced towards the cloud: applications are increasingly extracted from the personal computer and stored in a remote location to interact and share. One of the biggest advantages of the cloud is that it facilitates sharing, distributing creations among people. There is a similar concept to the one of sharing: **collaboration**, or being able to use the software simultaneously and in real time (Or almost real time) with other people and seeing the results. Wave embraced closely the concept of collaboration, and everything in it was geared towards it. Another successful Google collaborative product is Google Docs [1] that predates Wave, as a consequence since then there have been other collaborative applications like Zoho [2], Pads (Which are explained in Section 9.1) and more.

With the growth of Facebook they began increasing the variety of features, and one of them that relates to this work is the arrival of Facebook Apps: applications developed by third parties but integrated inside Facebook by using some of its features.

When Google announced Wave it was an innovative product, and attracted positive attention. Google, disappointed with the adoption Wave received,

and despite the enthusiasm from developers, stopped the development of Wave. They did not completely make it disappear, and instead chose to hand the project over to Apache to continue development, who deposited it on the Apache Incubator, a program for supporting promising open source software with the objective of converting it in an Apache Foundation project once it is mature enough.

Two of the important features of Wave since the beginning have been Robots and Gadgets: ways to **extend the capabilities of Wave** without the need to integrate them in the actual software, they can be made and changed independently. Gadgets are separate apps embedded inside another more powerful entity that hosts them. Robots interact with that same entity and are able to react to changes in it. The overall objective of this work is to create a framework around Wave, based on Gadgets and Robots, to explore their functionalities, help document their features and usage, and try to build a base so this two extensions can continue being explored and expanded in the future.

The P2Pvalue [3] project has inspired the existence of this work and what is needed to be done. The P2Pvalue is a "Techno-social platform for sustainable models and value generation in commons-based peer production in the Future Internet", and are studying different aspects of social networks, and building an online collaboration platform on top of Wave to promote "communities of collaborative production".

The aim of this work is a framework, understood as the structure supporting a conglomerate of elements with a similar purpose or context. When Google dropped support for Wave, they also began sparingly removing web pages with very important information about it, and some of that information has been lost.

1.1 Objectives

This end-of-degree project was not intended to be just another exercise, but rather accomplish actual contributions to a real existing project. It then tries to put together every piece regarding gadgets and robots, test everything relating to them and lay a rough path so it can be followed in the future. This involves **documenting and testing** all the features provided by the available APIs.

However that is not the only intention, it was also important to make some tangible **extensions as robots and gadgets** that are useful for the Wave community, that satisfy needs not yet met. Everything has been open sourced and released under a free license [4] so it can be further improved and analyzed. The project P2Pvalue [3] has been a great influence on what kind of features are needed, and some of them have been successfully accomplished.

- Wave Documentation
  - Improve existing documentation
  - Document undocumented features
- Projects
  - Updating existing projects
  - Generating new useful extensions
- Usage
  - Explaining undocumented procedures
  - Teaching how to start development of Wave extensions

1.2 Document Structure

All of those elements that compound the framework are explored individually. Those elements can be described individually, but the framework as a whole is also described. There are **global results and individual results**, and they are all organized as follows: first there is a global State of the Art and Methods, then each element is completely explained individually with its introduction, state of the art, results and conclusions, and finally global conclusions are reached. Also, every extension have their own results, conclusions and future work, but the complete work is analyzed as a single entity.

The objectives of the framework itself are the ones stated above, but every extension is also a whole project by themselves, so they have their specific objectives.

- Chapter 3 - State of the Art: alternatives to Wave gadgets and robots.

- Chapter 4 - Technologies and methods: every procedure and technology that has been used along this work.
- Chapter 5 - Frame for Development of Gadgets & Robots: how Gadgets and Robots work. How to get them running.
- Chapter 6 - CCWave: development of a Wave gadget for choosing a Creative Commons License.
- Chapter 7 - Pollymer: development of a Wave gadget for decision making.
- Chapter 8 - AppearWOW: development of a Wave gadget for video conference.
- Chapter 9 - Colbotia: development of a Wave robot for colorizing the contributions of each participant.
- Chapter 10 - Concluding Remarks: comments and analysis on the whole project.

Chapter 2

Introducción

Hoy en día las redes sociales en línea tales como Facebook o Google+ son de gran importancia en el uso convencional de Internet. Numerosas redes sociales han surgido en un corto período de tiempo, han sido diferentes e innovativas, y disfrutan de una creciente importancia fuera de la red. Uno de esos destacables intentos fue **Google Wave**, que tuvo una breve existencia antes de que Google decidiera dejar de dar soporte para él y desarrollarlo, e incluso desactivó los nodos que habían estado funcionando hasta entonces. El control del proyecto le fue otorgado a Apache para que continuara con su desarrollo. Desde el origen de Wave, la web ha sufrido numerosos cambios, pero la tecnología y el protocolo de Wave siguen hoy en día siendo relevantes. JavaScript es una de las tecnologías web más usadas hoy en día para ejecutar código en el cliente. El mundo ha girado en el sentido de "la nube": las aplicaciones cada vez más son extraídas del ordenador del usuario y almacenadas en una localización remota para interactuar con ellas y compartirlas. Una de las principales ventajas de "la nube" es que facilita el intercambio entre usuarios, distribuir creaciones entre otras personas. Existe un concepto similar al de intercambio: **colaboración**, o usar aplicaciones de manera simultánea con otras personas y en tiempo real (O tiempo casi real) para ver el resultado a la vez. Wave siempre mantuvo cerca el concepto de colaboración, y todo en él estaba diseñado para colaborar. Otro exitoso producto de Google relacionado con la colaboración es Google Docs [1], que antedata a Wave, y por ellos desde entonces han existido otras aplicaciones colaborativas como Zoho [2], Pads (Explicados en la Sección 9.1) y otros.

Acompañando al crecimiento de la red social Facebook, desarrollaron varias nuevas utilidades, y una de ellas está muy relacionada con este trabajo y son las Apps Facebook: aplicaciones desarrolladas por terceros e integradas dentro de Facebook usando algunas de sus características.

Cuando Google anunció Wave fue una innovación, y atrajo una atención

positiva. Google, decepcionados por una mala acogida del producto, a pesar de la buena aceptación por parte de los desarrolladores, decidieron ponerle fin al proyecto. No hicieron que desapareciera completamente, en su lugar eligieron otorgarle el proyecto a Apache con la intención de que continuara el desarrollo, y lo pusieron en el Apache Incubator, un programa para ayudar a prometedores proyectos de software libre con el objetivo de acabar incorporándolos como proyectos de la Apache Foundation cuando estén lo suficientemente maduros.

Dos de las características de Wave desde el principio han sido los Robots y los Gadgets: formas de **extender las capacidades de Wave** sin la necesidad de integrar nada en el software de Wave, así que pueden ser hechos y modificados de manera independiente. Los Gadgets son aplicaciones separadas pero empotradas dentro de una entidad más potente que ellos y que se encarga de contenerlos. Los Robots interactúan con esa misma entidad y pueden reaccionár a sus cambios y eventos. El objetivo general de este trabajo es crear un marco alrededor de Wave basado en Gadgets y Robots y así explorar sus funcionalidades, ayudar a documentar sus características y uso, además de cimientar una base para que estos dos tipos de extensiones puedan seguir siendo exploradas y mejoradas en un futuro.

El proyecto P2Pvalue [3] ha inspirado tanto la existencia de este trabajo así como lo que era necesario hacer. P2Pvalue es una "Plataforma tecno-social para modelos sostenibles y generación de valor en la producción de iguales basada en lo común en el Internet Futuro", están estudiando diferentes aspectos de las redes sociales, y construyendo una plataforma online colaborativa basada en Wave para promover "comunidades de producción colaborativa".

El objectivo de este proyecto es un framework, entendido como la estructura que soporta un conglomerado de elementos con una intención o contexto similares. Cuando Google dejó de dar soporte para Wave, también empezaron lentamente a eliminar páginas web con información muy importante sobre Wave, y partes de esa información se han perdido.

2.1 Objetivos

Para este proyecto de fin de grado estaba pensado que no fuera simplemente un ejercicio más, sino llevar a cabo contribuciones reales a proyectos existentes. Por lo tanto se unen todas las piezas relacionadas con los Gadgets y

Robots, se prueba todo lo que tenga que ver con ellos y se dibuja un camino que podrá ser seguido más adelante. Para ello hay que **documentar y probar** todas las características que se ponen a nuestra disposición gracias a todas las APIs.

De todas maneras ésa no es la única intención, también es importante hacer algunas aportaciones tangibles en forma de **extensiones como Robots y Gadgets** que sean útiles para la comunidad de Wave, y que satisfagan necesidadas existentes que todavía no han sido solventadas. Todo el proceso se ha liberado en código y otorgado una licencia libre [4] para que otras personas puedan analizarlo y mejorarlo. El proyecto P2Pvalue [3] ha sido una gran influencia en cuanto a qué características eran más necesitadas, y algunas de ellas han sido completadas satisfactoriamente.

- Documentación de Wave
  - Mejorar documentación existente
  - Documentar características no documentadas
- Proyectos
  - Actualizar proyectos ya existentes
  - Generar nuevas extensiones útiles
- Uso
  - Explicar procedimientos no documentados
  - Enseñar cómo introducirse al desarrollo de extensiones Wave

2.2 Estructura del Documento

Todos esos elementos que componen el framework serán explorados de manera individual. Pueden ser descritos individualmente, pero a su vez se puede describir el framework como unidad. Hay disponibles **resultados globales así como individuales**, y todos ellos están organizados de la siguiente manera: en primer lugar se encuentra un Estado del Arte general así como unos Métodos, y después cada elemento se explica completamente de manera indificual con su propia introducción, estado del arte, resultados y conclusiones. Finalmente se alcanzan unas conclusiones globales. Además de eso, cada

extensión tiene sus propios resultados, conclusiones y trabajo futuro, pero el trabajo completo es analizado también como una entidad individual.

Los objetivos del framework en sí mismo son los descritos arriba, pero todas las extensiones son un proyecto completo en ellas mismas, por lo que tienen objetivos específicos.

- Capítulo 3 - Estado del Arte: alternativas a los Robots y Gadgets de Wave.
- Capítulo 4 - Métodos y Tecnologías: cada procedimiento y tecnología que hayan sido usados a lo largo de este trabajo.
- Capítulo 5 - Marco para el Desarrollo de Gadgets y Robots: cómo funcionan los Gadgets y Robots. Cómo se pueden ejecutar.
- Capítulo 6 - CCWave: desarrollo de un gadget de Wave para elegir una licencia Creative Commons.
- Capítulo 7 - Pollymer: desarrollo de un gadget de Wave para toma de decisiones.
- Capítulo 8 - AppearWOW: desarrollo de un gadget de Wave para realizar vídeo conferencias.
- Capítulo 9 - Colbotia: desarrollo de un robot de Wave para colorear las contribuciones de los participantes.
- Capítulo 10 - Comentarios Conclusivos: comentarios y análisis del proyecto completo.

Chapter 3

State of the Art

This chapter will explore the global state of the art related to this work, and the evolution of gadgets and robots along the existence of the Internet.

3.1 Generic Overview

The concept of gadgets might have born with the name of apps in Google's "Google Personalized Homepage" in 2005 [5], later to be called iGoogle (Figure 3.1) and rename these components to gadgets. There is an important distinction to make: Nowadays hearing the words apps reminds us of portable phone applications. The word app insinuates smaller size and lesser complexity, and that is true for both kind of apps, but this work is exploring those **applications than run inside a bigger environment** and are embedded in it, extending it, but are not standalone and complete by themselves, they depend on the parent framework. The social aspect of this apps was being able to share your gadgets with other people, and some examples of them would be weather apps, stock apps or video-watching apps.

In the latest years of iGoogle's life, before being discounted in 2012, iGoogle apps coexisted with Google Wave [6], and Google Wave's Gadgets (Figure 3.2). This other gadgets offered the advantage of the Federated Wave Protocol [7] and **real-time collaboration** [8].

Then Google began pushing Google+ [9], which can be considered to have some kind of gadgets in the shape of games, which were heavily influenced by Facebook games [10], allowing users to play with other people from the same social networks, giving gifts and sharing scores.

Google also briefly had Spreadsheet Gadgets, which added unusual functionality for a spreadsheet application in Google Docs.

Figure 3.1: iGoogle in 2008

But Google has not been the only player in the world of applications. In the neat past Flash [11] components have been omnipresent, basically for games but not limited for them. Another alternative for applications in the web are Java Applets [12], commonly use for scientific and learning purposes.

The other aspect explored in this work are robots, also known as bots. Robots usually interact loosely coupled from the software they are meant to interact with, but as with Wave Gadgets, robots can be incorporated in the main software, acting as an extension to it. An example of this are uNrEaL Bot [13], a collection of robots for very popular video games created to give the user an advantage over the rest of the players, integrating the graphical interface inside the game, and altering the game's own engine.

Robots are also **important in the web world itself**, but they are not usually productive bots, but instead robots made to emulate human behaviour to trick systems and people and get some kind of profit from it. It has been estimated that one third of the world's Internet traffic is made by automated robots [14]. But this robots are different in the aspect that they are not

Figure 3.2: Google Wave Gadgets

meant to be: There is no intended resources for them and they take advantage of vulnerabilities, so as a result they can not run in harmony with the parent software, but as a parasite.

3.2 State of the Art for Extensions

As several extensions will been seen here, the specific state of the art concerning each of them has been explored deeper in their specific sections 6.1, 7.1, 8.1 and 9.1 respectively, so they are closer to their context.

Chapter 4

Technologies and Methods

This chapter will explain all of the technologies and techniques that have been used and applied to finish this work.

4.1 Technologies

Multiple utilities for a wide variety of purposes have been used. Some of them such as the choice of web server or Integrated Development Environment are alternatives selected for their personal perception of fitness to the task to solve. Others such as Wave and the Google Web toolkit are essential and irreplaceable.

4.1.1 Wave

Wave can be interpreted as different things: The project formerly known as Google Wave and now Apache Wave, the system Wave In A Box which is the main focus of the Apache Wave project, the communication protocol inside Wave In A Box created by Google for **human communication** and inspired in email, or one of the units of communication inside this protocol. In the making of this work every extension has been developed and tested under two varieties of Wave: Wave In A Box and Kune [15]. Kune nowadays has much of its code forked from the Wave In A Box repository, but adding more functionality around it or changing it. Gadgets and apps are built into the core of Wave, and, as such, Kune inherits all their capabilities, making the gadgets and robots explored here compatible with both technologies. The people behind Kune are Colectivo Comunes [16], a non-profit collective that tries to help other collectives to carry their operations by developing some web tools and free resources.

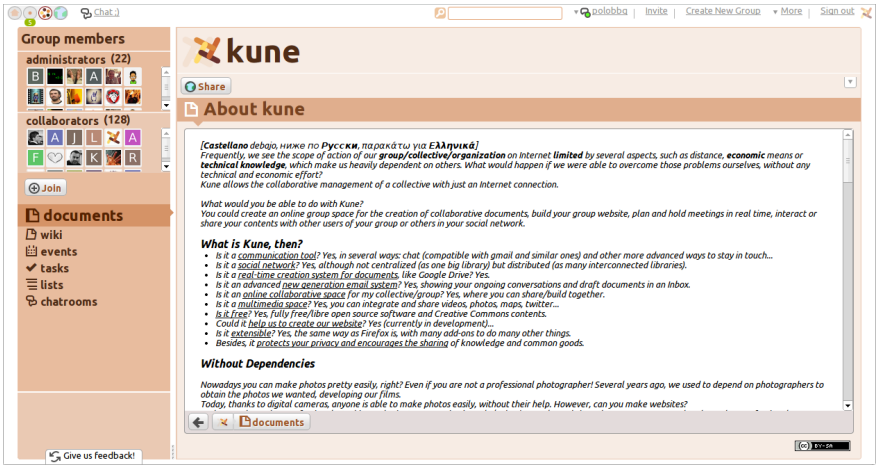

Figure 4.1: Kune Groups

The main technical feature that characterises Wave is the **Wave Federation Protocol** [17], that handles all of the communication happening between users. It is an extension of the Extensible Messaging and Presence Protocol [18] that adds federation on top of it. Wave being an open protocol allows anyone to be a Wave provider, Participants send delta changes on the content, and those deltas are distributed through the rest of the participants, guaranteeing that the end result is the same for everyone [19]. Gadgets and robots will work under this environment, interacting with the Wave protocol.

4.1.2 Google Web Toolkit

Google Web Toolkit [20], or GWT, is a set of tools that allows web developers to code in Java and from that java-compliant code, generate Asynchronous JavaScript and XML (AJAX) code to make front-end applications. The GWT framework focuses on efficiency and cross-browser compatibility, generating **AJAX from Java code** and then serving different compilations for every browser and locale combination, so the elements are rendered as they should in each browser, even though they behave differently. GWT provides the developer with all the common web controls, allows RPC invocations, browser history management, unit testing, and native JavaScript calls, among other features.

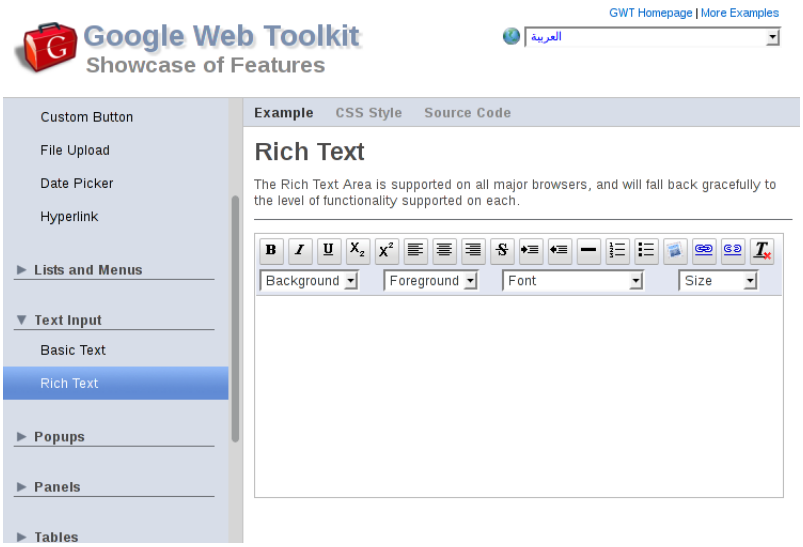

Figure 4.2: GWT Showcase

GWT has a development mode, where the Java code is not compiled to JavaScript, but instead it is ran in native Java and simulates the JavaScript components that will be used later. The development mode also allows a Java debugger to attach to GWT and debug the application locally.

4.1.3 Gadgets API

The Gadgets API [21] is made by Google to **embed third party applications inside various Google products**. They implemented it in wave as Wave Gadgets API, to have access to the specific features of Wave. These applications are HTML and JavaScript, with all their possibilities, and access specific features that the server can offer.

This is a Java API, and it needs to be compiled with the GWT compiler in order to be able to link to the gadgets. Apart from being able to use all GWT components, this API lets the developer communicate with Wave, and that means mainly altering the state and responding to state changes. The state contains the participants, and the content, plus more information.

4.1.4 Robots API

The Robots API [22] is an API with client libraries for Java an Python, both of them implementing the same set of features. The original intention was to make **robots able to act in Wave like human participants** and in exactly the same places and same ways as actual participants could. In practice, not all interactions are implemented for robots. A robot can be added to a Wave, can edit text, add participants, publish new content, and access the contents of all of them, as well as responding to events about changes in the document or participants.

4.1.5 Other Technologies

Also worth mentioning are the following technologies: As an IDE Eclipse [23] 3.8 has been used for developing the gadgets as well as the robot. OpenJDK [24] 7 has been used as Java Development Kit. Jetty [25] 6.1.24 has been used for a local robot server. Tomcat7 [26] along with Shindig have been used as a local gadget server. Chromium [27] 32 and Iceweasel [28] 24.3 have been used for testing the results. Maven [29] has been used for dependency management. The service Appear.in has been used to make the video interaction in the Video Conference Gadget. The Java library google-diff-match-patch has been used to calculate diffs between texts for creating Colbotia. Vaadin's GWT Graphics library has been used in the Decision Maker Gadget to draw some of its interface. Figure 4.3 shows the logos of most of the technologies used. Table 4.1 shows a selection of technologies along with their purpose.

Type	**Name**	**Use**
APIs	Robots API	Creating Java Robots for Wave
	Gadgets API	Creating Gadgets for Wave
Servers	Tomcat	Web server for storage
	Shindig	Gadgets server with OpenSocial support
	Jetty	Java server for robots
Wave	Wave In A Box	Testing Robots and Gadgets
	Kune	Testing Robots and Gadgets, documenting
Libraries	Appear.in	WebRTC video communication
	Google diff-match-patch	Calculating differences between two texts
	GWT Graphics	Drawing shapes in GWT

Table 4.1: Technologies Summary

Figure 4.3: Technologies Used

4.2 Methods

This chapter will explain the different techniques and decisions used and taken to successfully meet all the requirements and approach the problems trying to be as close to the software engineering process as possible. There has been a special focus on open source, standards and patterns.

4.2.1 Free Open Source Software Approach

Everything here such as this report, and the whole code has been **developed openly** and released under a relatively restrictive copyleft license. A Github Organization is a conglomerate of organised software repositories stored together and hosted on github.com. The progress, as well as the code, can be visited at the following Github Organization:

`https://github.com/End-of-degree-project`

The license chosen for the software is the **AGPL**, which is an extension of GPL. GPL basically affects distribution of derivative works, forcing the distributor to provide the source code of everything and distribute it with the same license. AGPL has the same restriction, but extends GPL by also forcing the code to be published under the same license, even if the derived work is not distributed.

It is important to observe that open source and open development are not related one to another. Although sources differ when defining what open source is, they all basically agree that source of the software should be available, and some liberties of use, modification and distribution are given to the users. This project is open source, but it has also been developed openly. That means the evolution of the work as well as the one of this document can be seen in the commits of the repositories. Collaboration would have also been accepted, and issues attended.

4.2.2 Standards & Protocols

Open source software and free protocols have been used when possible. That improves the visibility of the project and increases the chance of it being reused, modified, and useful even for the purpose of other people.

OpenSocial [30]: They efficiently describe themselves "OpenSocial Gadgets are a mechanism for embedding one web application into other web application using modern Javascript and HTML5 technology". They basically are a standard that make gadget implementation possible, it is a public specification of various procedures to create a "hosting environment", understanding that as a web container that can house other elements.

Wave Federation Protocol [7]: Again, a good definition is the one given to themselves as "a server-to-server network protocol between service providers, supporting low-latency, concurrent updates to conversations (live typing) and domain authentication". But the most relevant aspect for Wave is that it provides federation, which again in their own words is "The utility of waves is greatly enhanced if they can be federated in the sense that they are shared between users from different organisations, hosted by different service providers across the Internet".

Much more information can be found on the Google Wave Federation White Paper [31] and the Draft Protocol Specification [32].

Also, the structures have been planned and modelled beforehand in UML [33] in order to be implemented later. **UML diagrams** can be found in the Table of Figures with the prefix `UML`.

4.2.3 Patterns

The Model View Controller pattern has been followed in every gadget when interacting with with the GWT components. The Model View Controller makes use of object orientation to divide a software application to separate the internal representation of the software from the way it is shown to the user.

The Observer pattern has been implemented in all the gadgets in order to wait and answer to state changes on the Wave's state. In this pattern an object, called the subject, holds a list of other dependent classes called observers. When a state change that wants to be notified happens on the subject, it will notify the observers of the change.

Use of the Dependency Injector has been carried out by using GIN in both the gadgets and the robot to satisfy dependencies in an inverse order [34]. This pattern implements the dependency inversion principle, to make high-level modules not dependent on low-level elements, exactly the opposite as the traditional dependency pattern. To achieve it the dependencies are passed by reference to the dependent object. Using GIN or Guice makes this process very simple.

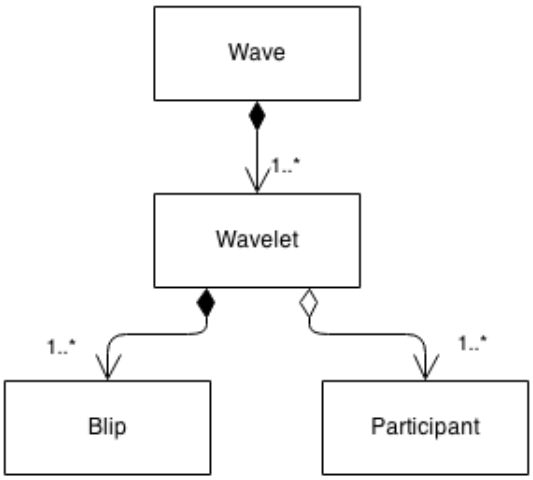

Figure 4.4: UML Class Diagram, Wave Structure

4.2.4 Extensions

Both possible ways for extending Wave have been explored in this work. Gadgets have been implemented with the Wave Gadgets API, and robots with the Java version of the robots API.

To understand where these extensions live, it is necessary to understand how wave is structured. In Figure 4.4 you can see **how Wave documents are structured**. The outermost layer is the wave, a thread or conversation, the whole picture of a conversation. Users are invited to the wave, letting them participate in any of the inner wavelets, when they then become participants. As of today, in current versions of Wave In A Box and Kune, there is only one wavelet inside every wave, but the Wave Protocol allows for multiple wavelets. **Blips are individual messages created by a participant**, and edited by participants later. Blips have hierarchical structure and can be nested inside other blips. Blips also contain a document in XML-like format, which is the text itself plus all the variations that Wave supports such as annotations.

Chapter 5

Frame for Development of Gadgets & Robots

As stated before, there is an important lack of documentation affecting anything relating to Wave. Through the progression of this project some obstacles, and the following chapter describes how they can be overcome. In addition to that, it explains the structure followed to implement the extensions as well as the operation of part of the features in Wave.

5.1 Gadgets

As explained before, gadgets are applications by themselves, but depending on some bigger entity, in this case the Wave server. Gadgets are inserted inside a document, together with the content itself. They are **placed inside a blip**, but belong to the wave and therefore have access to the whole wave's content, which is given access by the Gadgets API. But that access is very limited, almost only to access what is called the wave's state, a key-value dictionary that keeps track of every state change that occurs inside the wave.

There is actually two differentiated kind of states in a wave:

- **Private State**: stores information that can only be accessed and modified from the participant that created it. Useful to keep stored private information that is not intended to be shared.
- **Shared State**: stores the global state of every gadget in this wave, representing the whole picture. Useful for communicating with the other participants and collaborating or communicating in the gadget.

It is important to know that the state belongs to the wave, so gadgets will have access to the other gadgets state, even different instances of the same

gadgets. It is then good practice to adopt a system similar to namespaces in programming languages, appending an identifier of the gadget before the key in a way we are not altering or reading an unintended state.

Each actual instance of the gadget in a participant's browser is local, **variables are not shared** in any way between different participants. When a user triggers a state change in his local instance of the gadget, the only thing that can be communicated outside of the browser is a delta, that is an addition, deletion or modification in the key-value pairs that represent the state. This delta is then communicated to the local instance of the gadget in every client's browser so they can act accordingly and represent the new state. That behaviour can be seen illustrated in figure 5.1.

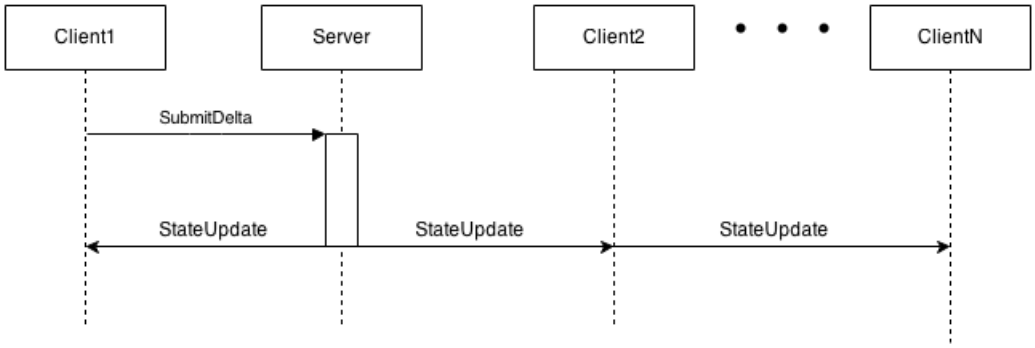

Figure 5.1: UML Sequence Diagram, Wave state transmission

The Gadgets API asks you to follow a class hierarchy in order to be able to correctly communicate with wave. That hierarchy is as seen in Figure 5.2.

Those classes will interact in the following way:

- AbstractGinModule: GIN is Guice for GWT client-side code, built on top of Guice and with a subset of Guice binding language. It allows the programmer to follow the pattern of dependency injection. This module attaches all of the modules together.

- Composite: Is the actual visual representation of the gadget. This is a GWT class ideated for creating custom widgets. Composites can contain a panel, and inside the panel complex GWT component hier-

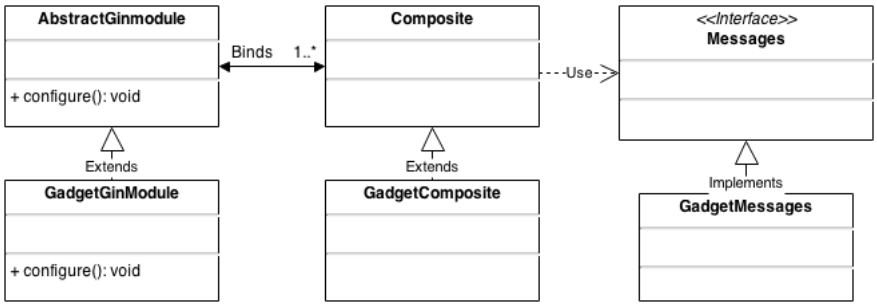

Figure 5.2: UML Class Diagram, Gadgets API class structure

archies can be built. All features of GWT are supported thanks to this component.

- Messages: A GWT interface meant to be implemented by a class that gives access to the strings needed to build the interface. Those strings can be localised to make the gadget available in different languages for each client. The locale is determined by the Accept-Language field in the browser's HTTP request [35] and served at runtime to the browser. Typically this class will be used from the Composite, or any of its inner components, to fill them with human-readable language.

With every one of the gadgets there is available a **tester and a deployer**, both of them depending on the main gadget, which is common for both.

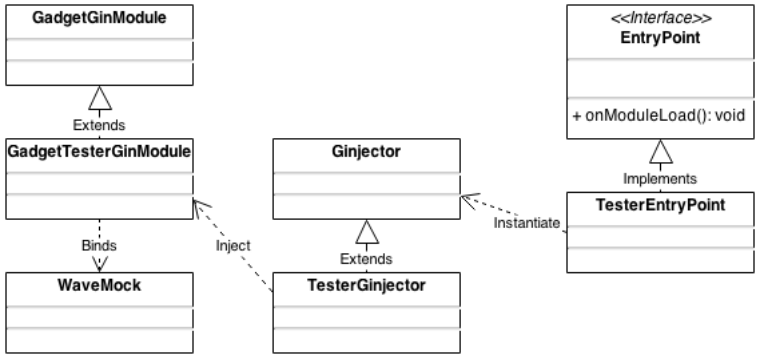

Figure 5.3: UML Class Diagram, Gadget Tester Structure

The tester project seen in Figure 5.3 shows the common class hierarchy used in every one of the **tester projects**, which makes use of the development

mode available in GWT, so gadgets can be quickly tested before deploying them. It is linked to the one seen in Figure 5.2 by the GadgetGinModule, which is inherited by the GadgetTesterGinModule. This new Gin module adds another injection, the WaveMock. This is a mock class that **simulates the basics of the behaviour of the Wave infrastructure** without the need to deploy the whole Wave system. To complete the needs of GIN a class extending Ginjector is created, the TesterGinjector, whose task is to actually inject the GinModule. This injector is instantiated inside a class that implements the interface EntryPoint that, as its name says, will be the entry point of our gadget in the onModuleLoad method. This project has to be run as a Web Application with Google's App Engine.

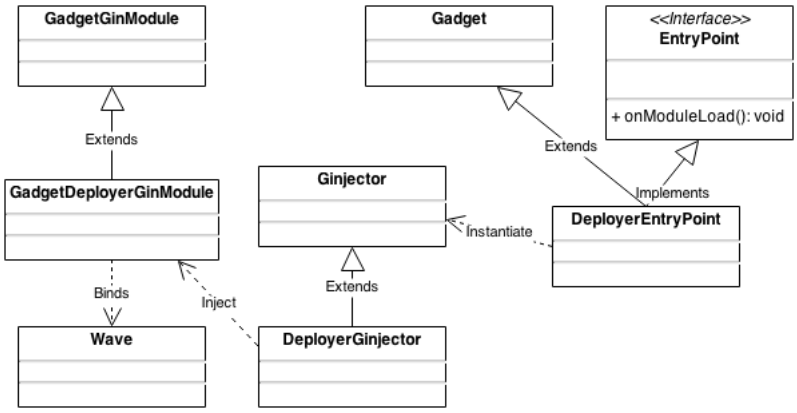

Figure 5.4: UML Class Diagram, Gadget Deployer Structure

The **deployer project** represented in Figure 5.4 is the project used to generate all the final JavaScript files ready to deploy to a web server with support for gadgets. An intermediate XML file containing JavaScript code is generated with enough logic to figure out which one of the browser and locale configuration has to be served.

It is very similar to the tester organization but with two basic differences. First, the GadgetGinModule now binds the class Wave instead of a Wave mock, as it will be running in the **whole Wave environment**. The other difference is that the EntryPoint, as well as implementing the same interface as before, now also extends the Gadget class, coming from the Gadgets API and representing a gadget. This project needs to be GWT compiled to successfully generate all the files.

Gadgets have been tested for deployment using Tomcat as a web server and Shindig as a gadgets server with support for OpenSocial. Several files are generated after GWT compiling, and they should all be put together in the same directory and accessible from outside the own server. Shindig is then deployed as an application inside Tomcat.

Shindig needs to be configured accordingly:

- First add "wave" to the gadgets.container in the `container.js` file, in order to support Wave. As Wave does not use security token (OAuth's authentication token) for gadgets, you have to set Shindig not to require it by adding the following line: `''render_token_required'' : false,`.
- Then you have to instruct Wave to use the gadgets server that was configured. The place to do that is a file called server.config in Wave In A Box, and wave-server.properties in Kune. Set the properties `gadget_server_hostname` to the host running Shindig, and `gadget_server_port` to the port where it is running.

5.2 Robots

Robots were created in Wave with the intention to be able to act **exactly as an actual human participant**. Revisiting Figure 4.4 we can understand how robots can participate: They are invited to the Wave by inserting their unique identifier equivalent to the identifier of actual participants. They will appear as invited without any apparent indication that they are a robot. Robots can make changes to the wave such as creating a new blip, editing them, changing annotations, among other.

In order to be able to interact with Wave it necessary to register the robot

Figure 5.5: Robot Registration Screen

with the Wave server where the robot will be run. For example, for the Kune node kune.cc, go to `http://kune.cc/robot/register/create` and you will

be prompted with the screen shown in Figure 5.5.

As username set the name you wish to give your robot, it should be a free name in the server, as names are unique. In the URL enter the URL where the gadget will be launched, it should be an URL reachable from within the Wave server. After sending the data you will receive a Consumer Token matching the name you entered, and a Consumer Token Secret: a secret key you will need to use to authenticate with OAuth and guarantee the identity of your robot.

There is no equivalent to the tester mode from GWT, but running a server for the robots is easy enough so that it is not a hassle. Using Maven plus Jetty in eclipse you can set the Maven goal to `jetty:run` and the robot will be run.

Robots do basically two things: **Act when needed, and react to events**. Acting means modifying the documents, creating new blips, and more actions. And receiving events means a callback will be made to the robot when something from an external action happens on the Wave. The kind of events that are notified to the robot are the following:

- WaveletBlipCreated: Triggered when a new blip is created.
- WaveletBlipRemoved: Triggered when a blip is deleted.
- WaveletParticipantsChanged: Triggered when a participant is added or removed.
- WaveletSelfAdded: Triggered when the own robot is added as a participant.
- WaveletSelfRemoved: Triggered when the own robot is removed as a participant.
- DocumentChanged: Triggered when the text of a document changes.
- AnnotatedTextChanged: Triggered when the annotations of a document change.

They are not compulsory but optional, **the developer can choose which events** will be transmitted. A public file called "capabilities.xml" has to be created in order to notify the Wave server of which events are implemented,

and a hash is calculated based on those events to make it easily visible when changes in how the robot acts have been made, because the hash will change.

There are other events documented such as GadgetStateChanged or WaveletTagsChanged, which even though available through the Robots API will not be triggered on the server, and therefore never received.

Chapter 6

CCWave: A Wave Gadget

As it can be seen in the Design Guidelines of the P2Pvalue project, in the Legal Regime section [3], they identify software licenses as a key feature: "We should secure that none of the content produced / uploaded on the platform is infringing: Draft specific terms of use (if we use the Wikipedia model, rather than the Kune model) or include a mention during sign-up process for contributors to accept to **release their work/data under a particular license**" and state the importance of keeping information free "A large number of respondents described that the **commercialization of the commons by third parties would reduce their motivation to contribute**. Provide guidance mechanism to help users choose between different licensing schemes based on the interests of the commoners involved in the project".

The default license for any creation of any kind is the restrictive copyright. Copyright tries to ensure that all the rights remain to the original author of the content, but sometimes it can be advantageous to let people freely or semi-freely use, distribute or modify the content [36]. **One of the most popular of these licenses**, with over 400 million [37] licensed works, are the Creative Commons set of licenses. These are not adequate for licensing software, but in the context of Wave the content is usually creative text work, for which Creative Commons suits perfectly.

Thanks to the development of this Gadget, it has been documented in the P2Pvalue wiki's article regarding Gadget Development the need for an OpenSocial server in order to run Gadgets [38].

6.1 State of the Art

Wave does not have any way of publishing your content under any specific license. Kune has the option for publishing Waves in your personal space

under one of the existing Creative Commons Licenses. Outside of the personal space, Kune preserves Wave's aspect and removes the space where the license is in the personal space, so no license is imposed to other participants in a wave.

Outside the world of wave, there's similar alternatives to the one trying to be implemented here. There is for example Creative Commons Configurator for Wordpress, which lets you set a license for your Wordpress posts. Creative commons also has a license chooser [39] in their own webpage, that selects a license based on questions asked to learn the needs of the user.

There is a limited amount of Creative Commons licenses, all of them requiring attribution of the original creator. There is some concepts which can be combined to build the different licenses:

- Attribution: The used of this work has to be attributed to the original author and distributed with this attribution. If the BY is present, attribution is required.
- Derivation: Derivative works are those based on a previous work and modified to make a new work. If the ND is present, no derived works are allowed.
- Commercial: A commercial work is that which seeks a profit directly or indirectly. If the NC is present, no commercial use of this work is allowed.
- Share Alike: Share the work the same way it was previously shared. If SA is present, usage of this work is forced to be shared alike.

Not all combinations of this concepts are possible, Table 6.1 shows the existing licenses and their denominations.

CC-BY	Attribution
CC-BY-SA	Attribution + Share Alike
CC-BY-NC	Attribution + Non Commercial
CC-BY-NC-SA	Attribution + Non Commercial + Share Alike
CC-BY-ND	Attribution + No Derivatives
CC-BY-NC-ND	Attribution + Non Commercial + No derivatives

Table 6.1: Available Creative Commons licenses

6.2 Results

This extension allows participants to **set a Creative Commons license** to a specific blip, by inserting a gadget in the content and answering questions to reach the adequate license that meets the requirements. The Figure 6.1 shows the final result of this extension.

Relating to the gadget structure described in Figure 5.2, this Gadget's Composite is the class CCGadgetMainPanel, the Messages relates to CCGadgetMessages and the GinModule would be represented by the class CCGadgetGinModule.

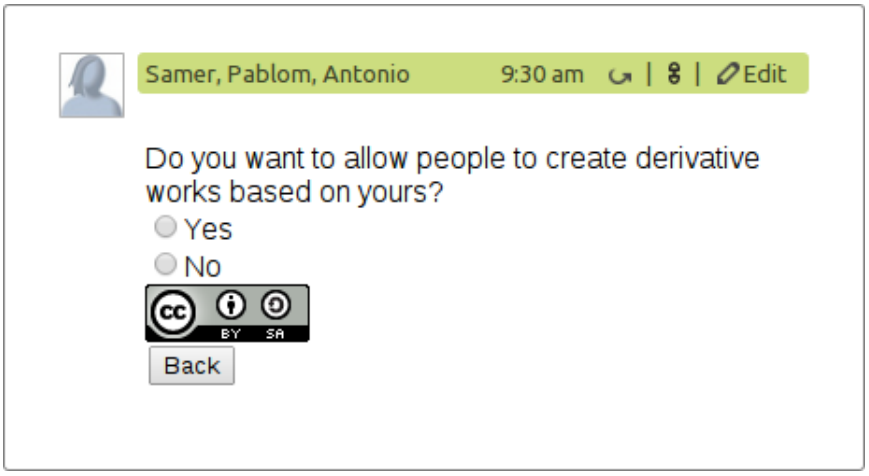

Figure 6.1: CCWave

This gadgets keeps a variable for each one of the possible concepts for a unique Creative Commons License, such as "Commercial use", and **asks a question to the user** to determine if the restriction has to happen or not, in order to figure out a valid license. The state stores the state of all the answers to the questions that can be asked, and the answers can be yes, no, or unanswered. When a question is answered a state delta is sent to the rest of the participants, and the image of the license is update to the best match from all the possible licenses. When all the answers are answered, only the image of the license remains linking to the whole chosen license.

6.3 Conclusions and Future Work

This extension is a Gadget that can be inserted in a blip to let the user select a Creative Commons license for the content of the blip.

The main problem with this extension is the **limited amount of licenses**, only Creative Commons. Even though they are widely used, they might not cater to the tastes of everyone, or fit every possible position. There is also no way to license anything other than a blip, such as a blip and all its children or a whole wave.

Those are limitations that can be solved starting with this extension and adding the possibility for more or different licenses. Solving the problem of licenses only being in a blip could not be done with Gadgets as they are by definition inside a blip, it would be compulsory to find a different solution.

Chapter 7

Pollymer: A Wave Gadget

The Design Guidelines for P2Pvalue [3] claim that some systems would be needed: "**System of polls / surveys** to engage users and facilitate administration/governance".

When talking in a group of people it is sometimes necessary to take a decision about a specific aspect. The easiest way to achieve this is by talking, and Wave with real time communication and editing common documents makes this easy. But this way it might be difficult to easily differentiate all the different options, see who agrees with each one of the options, or quickly decide one. This extension allows every participant to **choose an option or add new options to the decision**. Participants are also able to quickly see who voted each option along with their user avatar images. This features accompanied by Wave's style of communication make consensus and decision making easier.

Thanks to the development of this gadget an addition has been made to the P2Pvalue wiki regarding the dynamic height property of gadgets [38].

7.1 State of the Art

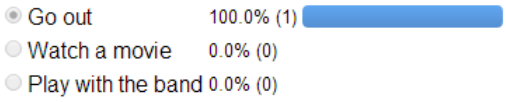

Figure 7.1: Poll

Because of the nature of Wave, decision making is a very important and basic, so there has been several gadgets made regarding that. It also easily

explores all the basic possibilities that the Gadgets API provides. Figures 7.1 and 7.2 show two of them: The first one is Poll by Eric Williams, focused on showing statistics about the results, maybe useful when there is a high amount of votes. The second one is Consensuall [40] by Antonio Tenorio, focused on sharing the personal opinion of each voter about an issue and reaching consensus on it.

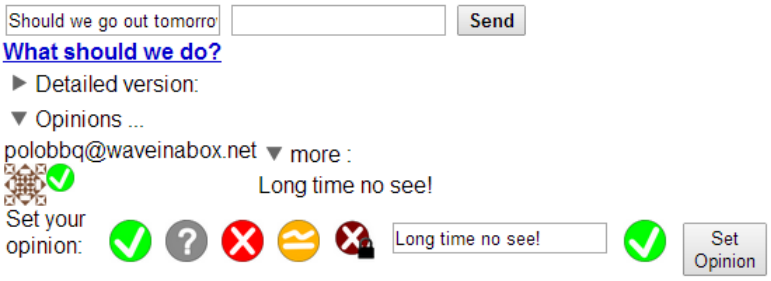

Figure 7.2: Consensuall

Pollymer falls in between both of them by letting people add their own answers, see a representation of the votes and who personally voted for each.

7.2 Results

Pollymer is a gadget that can be inserted anywhere in a blip. A title is chosen for the issue at hand, and then **different options** answering that title can be inserted by anyone who sees the gadget, not only the creator. One of the decisions can be instantly chosen by clicking on it.

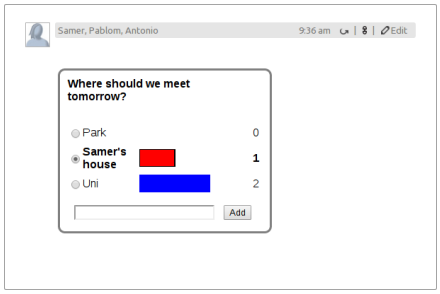

Figure 7.3: Pollymer

When there have been decisions chosen by different participants, **hovering the mouse over** the number of votes on the right will show the names and pictures of the participants that voted for it, as represented in Figure 7.5. Names and pictures are taken from their Wave profile.

This gadget also relates to the structure shown in Figure 5.2. The GinModule is the DecisionMakerGinModule, the Composite is the DecisionMakerMainPanel, and the Messages class is represented by the class DecisionMakerMessages. This gadget is slightly more complex than the Creative Commons one, so the class diagram with its specifics is shown in Figure 7.4.

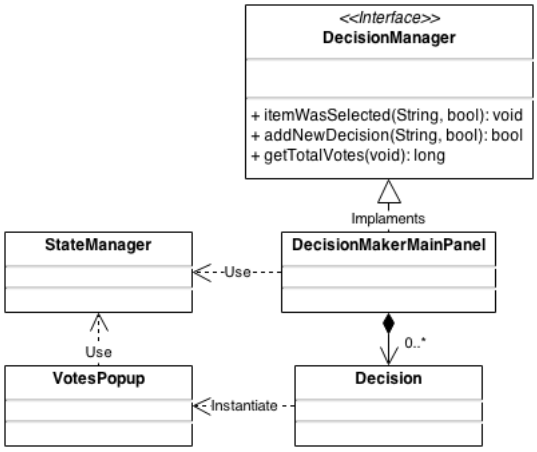

Figure 7.4: UML Class Diagram, Pollymer

The interface DecisionManager is extended. It is meant to represent all that can be done with the decisions: select one, add a new decision, and get the total votes of one specific decision. The DecisionMakerMainPanel implements it, and is also the container for all the decisions. A decision represents one of the options that can be chosen, and generates the Pop-up showing who voted for it. A StateManager is also used to handle everything related to the Wave's state.

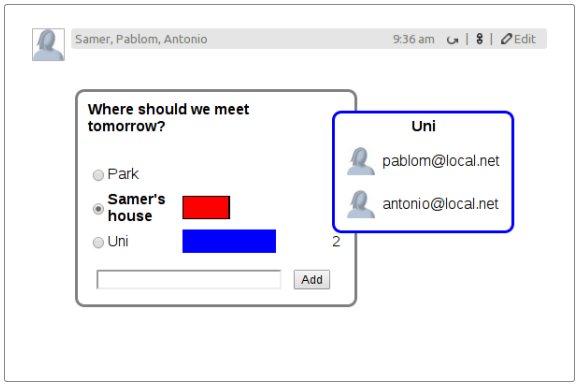

Figure 7.5: Pollymer Voters

There are three different types of entries in the Wave's state:

- **Count of votes**: There is one entry of this kind for each decision. To identify which decision this entry is for, the title of the decision is stored as a key. Therefore, the decision titles have to be unique. The value of this entry is the amount of votes that decision has, making it quick to retrieve the amount and show the vote count.

- **Voters**: Again, one entry for each decision and the title of the decision as an unique identifier. The value is a list of all the people that voted for that decision. As the state is only able to store a string, the list is like `|user1@kune.cc|user2@kune.cc|user3@kune.cc|`. This entry is used to know if a particular user has voted for a decision, and to fill the information inside the votes pop-up.

- **Title**: There is one single entry for each gadget. The value of this entry is the name of the decision to be taken. Used to fill the title after it has been set.

7.3 Conclusions and Future Work

This extension is a Gadget meant for voting on a statement, and seeing the opinion of the participants towards that statement.

There are already several alternatives for decision making, each one with their unique way of doing things, and there is certainly already one suitable

for every need, and Pollymer doesn't innovate in almost any aspect. This alternative has limitations though: There is no option for multi-choice answers, no way of editing the title after being created, and the list of votes can get a little uncomfortable to see after a large amount of votes has been made. Multi-choice answers would put this gadget **closer to a consensus tool**, and being able to block options would make the divergence among opinions more visible.

Chapter 8

AppearWOW: A Wave Gadget

Design Guidelines report of the P2P value project claims that "The platform should therefore include tools for collaboration, including key features, such as: Synchronous communication", among these synchronous tools they mention "Video and voice communication" [3].

Wave is meant for text communication. Text has the advantage that it can be stored easily, searched later and edited simultaneously, but lacks the naturallity of human-to-human interaction. To get as close as possible to it we need voice and video. Then collaborating on text can be made more efficiently. This gadget makes it possible to **speak to up to 16 other users and see them**.

8.1 State of the Art

There is no alternative to video or audio communication integrated on Wave. There is other alternatives though outside of it, some of them shown in Figure 8.1. They focus on video, even hiding text communication to leave more room to the video. AppearWOW can be mixed with text above and below, leaving the video communication as an addition and not the main point. Both video communication tools shown (Google's Hangouts and Microsoft's Skype) require you to have a specific user account to use their services, while this gadget lets you join the **video communication from within Wave**, but also from an external link without giving any kind of personal information.

Figure 8.1: Hangouts and Skype

8.2 Results

To make this gadget it has been essential the use of a pre-existing service called **appear.in** [41]. They provide the whole video and audio communication based on WebRTC [42], and also facilitate an easy way to use their service in an external website. It is a JavaScript component that can be put inside an iframe, and it will take care of almost everything. Figure 8.3 shows the final result of this integration. When a user inserts the gadget, he will

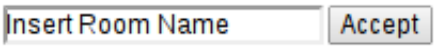

Figure 8.2: Room Selection Screen

be prompted with the room selection screen shown in Figure 8.2, letting him choose the identifier of the room people will meet in. The same room can be visited from different waves, and also directly from appear.in, as room names can not be duplicated. It was shown in Figure 5.2 the basic structure of a gadget. The composite is represented by the VideoGadgetMainPanel, the Messages are the class VideoGadgetMessages, and the GinModule is realised by VideoGadgetGinModule. Outside from that, the structure is relatively simple.

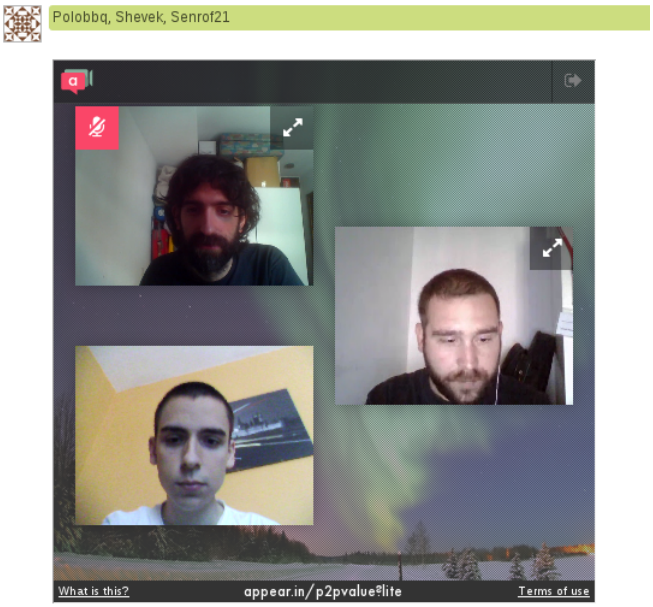

Figure 8.3: Video Conference Gadget

As appear.in is a JavaScript service, multiple calls to **native JavaScript** have been made inside this gadget. The service appear.in uses the name of the room as a unique identifier, so the gadget asks the user for a name before entering the room. The gadget also makes use of appear.in's `?lite` feature, that simplifies the user interface, leaving more space for the images of the video. The **camera is accessed through the browser**, so no additional software has to be installed.

The Wave state in this gadget is really simple, the only thing stored in it is the name of the room to enter it directly if it has already been set.

Figure 8.4 shows how the gadget reacts to requests. Once the gadget is inserted, the room selection screen will be shown, allowing any participant to select the name of the room the participants will meet in. When a room is selected, the participants automatically enter the room. If another participant joins the wave once the room has been selected, he will be served the video conference directly.

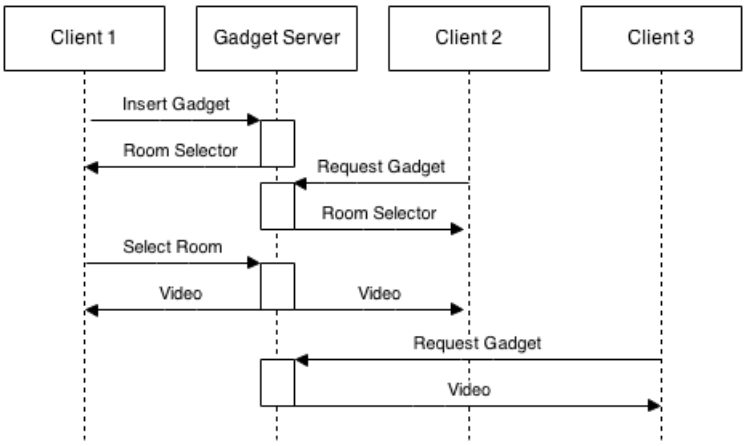

Figure 8.4: UML Sequence Diagram, AppearWOW

8.3 Conclusions and Future Work

This Wave extension is a Gadget which allows all the participants who are in the wave to join a video conference among them.

The fact that this extension is completely **dependant on an external closed-source service** is a big limitation. The service could stop working at any time, technical improvements on video or audio communication can not be made, limitations like the maximum of 16 online users can not be avoided, the way to use it could change making it necessary to update the gadget, and other changes may arise.

To solve all of these problems, an open source alternative for the video and audio communication should be used, or it should be developed from scratch.

Chapter 9

Colbotia: A Wave Robot

The Design Guidelines for the P2Pvalue project [3] talks about identifying the contributions from users: "The tool should have a way to capture contributions (such as open value accounting) and automatically distribute the monetary gains to the members based on their contribution".

Again, text communications gets in the way of collaboration in some aspects. When working between several people it is sometimes necessary to talk about some aspect of the work they disagree in. In plain text, when there are more than two collaborators, there is no **way to know who edited what**, so those kind of issues can not be addressed personally. A robot is needed in order to know when and what changes are being made.

Thanks to the development of this robot, the P2Pvalue wiki section documenting the Robot Development has been reviewed [43]. The README for the Robot Sample in P2Pvalue has been reviewed [44]. The Ŕobot Registration Process has also been reviewed [45].

9.1 State of the Art

In Wave it is possible to see who has participated in a blip as shown in Figure 9.1, but it is **not possible to see exactly what that participant has modified** what part of the document.

Figure 9.1: Blip Participants in Apache Wave

The other thing Wave does to try to keep people informed on when changes happen, is to highlight the changes that just happened and write the author's

name next to it, as shown in Figure 9.2. The problem is it is not permanent, so you only realise of the change if you were already looking at the content being changed.

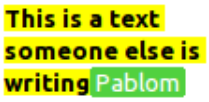

Figure 9.2: Change Highlighting in Apache Wave

Also, this extension is heavily inspired in Pads, services like PiratePad, Etherpad, TitanPad and many others, that offer an online collaboration tool that allows to concurrently write plain-text documents. They show a specific colour for each participant to quickly see who edited what. TitanPad in Figure 9.3, even though not the only, has the capability to show a timeline and revisit past states of the Pad, so no information is lost even after being modified.

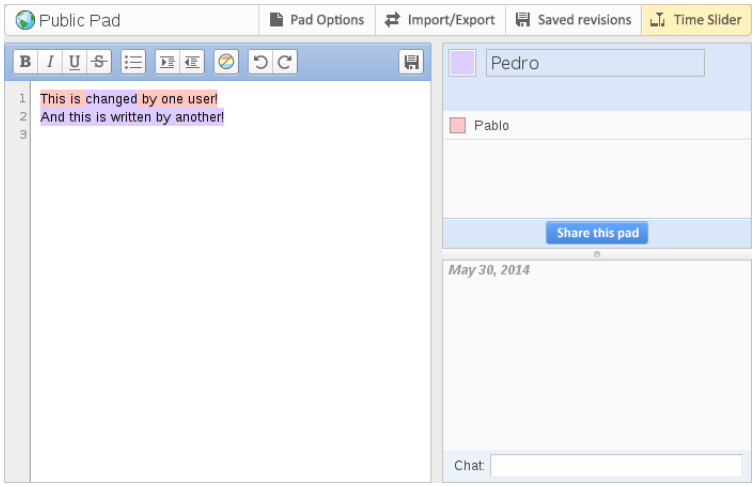

Figure 9.3: TitanPad

9.2 Results

This extension, in the shape of a robot, goes around that problem by **assigning a colour to each participant**, and painting the background of the text

that participant edits. The result can be seen in Figure 9.4. It also keeps track of who has each colour and puts it in a blip under the main blip of the wave, as seen in Figure 9.5. It is also possible to get around the colourising of any given blip by starting it with `@Robot clear annotations`, being "Robot" the actual name of the robot that it was registered with.

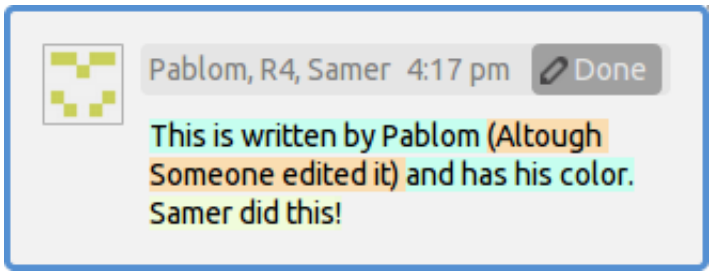

Figure 9.4: Colbotia Colours

The way to make the background of the text be of a specific colour is by changing the annotations of the document. Annotations in Wave are tags affecting a range of text and altering its properties, but they do not affect the text itself. Annotations are also able to be transmitted through the Federation Protocol. Every annotation is defined by a name (Specifying its purpose), a range (Determining the range of characters of text it affects), and a value (The value should that annotation take on that range). There are annotations for text size, links, language, among other. For this robot specifically the annotation **style/backgroundColor** is the one being set in the changed text.

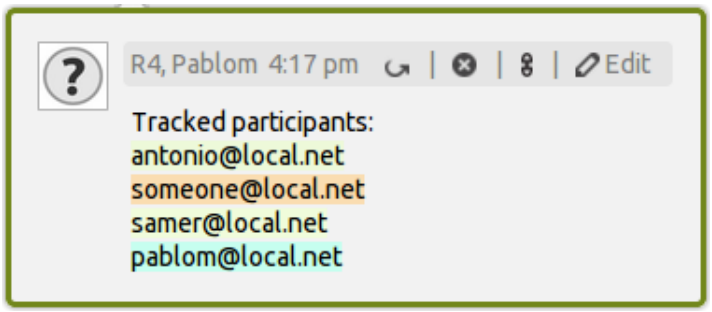

Figure 9.5: Colbotia Tracking Participants

The way to know when the changes happen is by subscribing the **DocumentChanged event** explained before. This event will be triggered anytime anyone modifies the text.

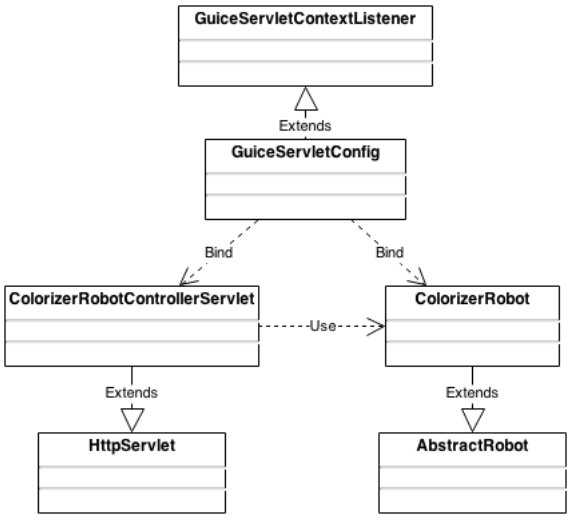

Figure 9.6: UML Class Diagram, Colbotia

That only leaves one task left: The DocumentChanged event tells us the document has changed, but not **what part of the document changed** or how it changed, so it is up to us to extract that information. To achieve it Google's google-diff-match-patch, a Java library that is able to extract the difference between two chunks of plain text. Every time the document is changed, the difference with the last version is calculated, and the new content is attributed to the participant that changed it.

Figure 9.6 represents the class structure of the Colorizer Robot. The HttpServlet is the responsible of receiving the communication from the Wave Protocol. The ColorizerRobot sets up the OAuth authentication with the token received when registering the robot. Also, thanks to the AbstractGadget class it is registered to receive all the events. The GuiceServletConfig binds the necessary dependencies together. Figure 9.7 shows the behaviour of this robot in time, disregarding the aspects of the robot registration. Once the server is started, it is ready to be used as a robot and **added as a participant to the wave**. When it is added, the first thing done is creating the

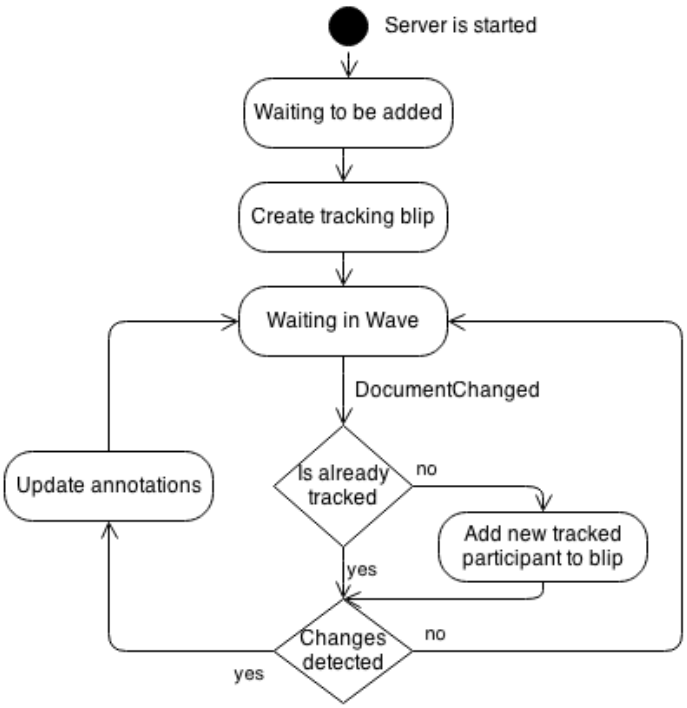

Figure 9.7: UML Activity Diagram, Colbotia

blip for tracking the participants that have acted. When the blip is created, it will start listening for Document Changed events. When each of those events is received, the robot will check if the participant that generated the event is already being tracked, and add it otherwise. The Document Changed event is triggered also when the document is changed by an equivalent one, so the difference of both documents has to be calculated. Annotations are updated according to the participants and changes. Afterwards, it goes back to waiting to another Document Changed event.

9.3 Conclusions and Future Work

This extension is a Wave Robot that can be added to a wave as a participant and will highlight in different colours all of the different participants to attribute edition of text to the author of that edition.

This extension, as said before, uses an API for diff calculation that works

with plain text. That means changes on annotations are not detected. Also, diff detection can be problematic in some specific cases. The difference between two documents is always computed correctly, but as the library does not have information on where the text changed, and it is not possible to extract that information from within the robots API, it will sometimes attribute a change to a wrong position of the text. For example: We have the string "abc", and a participant decides to add a new "b" to the left of the already existing "b". The end result would be "abbc", where the first apparition of the letter will be the change from the last version of the text. But because of how the diff calculation is made, the library will come up with the result that the difference between those two texts is the "b" on the right and the robot will put the background colour in the wrong character. This small example can be extrapolated to more complex and problematic cases.

Wave, thanks to the federation protocol and the deltas, is able to keep track of all the history of the modifications in the document. Also because of deltas, changes can be attributed to a participant with little overhead, as opposed to calculating the difference every time a change is made. With past states it would be possible to reliably go back in history and review what happened. **Implementing this feature at a Wave protocol level** would definitely be advantageous.

Chapter 10

Concluding Remarks

The whole end-of-degree project is analysed in the following sections regarding the results of the work, the conclusion reached after its development, and future lines of work that could be followed after making this work.

10.1 Discussion of Overall Results

Developing for Wave is **full of difficulties**. When Google released the project to the Apache Foundation to put into the Incubator they didn't fully release everything, keeping parts of the software hidden. There was lack of documentation for many features, and over time links with important information about Wave have been deleted without replacement. This informality leaves the developer with the only option of trial and error to make things work.

Many of these extensions, specifically the Creative Commons License Gadget, the Colorizer Robot and the Video Conference Gadget, or better said the base concepts behind them, had already been suggested for the P2Pvalue project to build, so they meet actual needs that the Wave community has right now.

There are attempts to **increase the accessibility to Wave** not only to users, but also to developers. That means improving visibility of all the projects related to Wave, completing the features that Wave is lacking right now, and documenting important things that act as tall walls for a developer that wants to get into it. This end-of-degree project contributed to review documentation about Wave gadgets and robots and add to it, as well as updating the sample gadget provided in the Kune repository to one working for the latest version of Kune and a newer version of the GWT.

It would have been useful to make a **bigger framework**, with more variety

and amount of gadgets. Suggested future work in the previous chapters is sometimes outside of the purpose of a framework for gadgets and robots, but the application framework itself can also be improved: Because of the increased difficulty in getting into the development there are procedures that were intended to be documented but haven't been. Also, this work is only as useful as the people who see it, then it will benefit from being spread.

The reasons for developing each of the extensions, as well as the contributions to documentation have been explained in the chapter for each extension, but Table 10.1 shows them summarised.

Extension	**Need Satisfied**	**Features**	**Interacts With**	**Contributions**
6 - CCWave	D1.3 Design Guidelines (Section Legal Regime) [3]	Choose a Creative Commons License for the content of a blip	Gadgets API	Gadget Development Tutorial [38], CCWave [46]
7 - Pollymer	D1.3 Design Guidelines (Section Technical Features) [3])	Poll users about a question	Gadgets API	Gadget Development Tutorial [38], Pollymer [47]
8 - AppearWOW	D1.3 Design Guidelines (Section Technical Features) [3]	Video conference communication	Gadgets API, appear.in	AppearWOW [48]
9 - Colbotia	D1.3 Design Guidelines (Section Economic Model) [3]	Highlight the contributions of each participant	Robots API, OAuth	Development Tutorial [43], Sample README [44], Registration Process [45], Colbotia [49]

Table 10.1: Extensions Summary

10.2 Conclusion

Gadgets by nature are just embedded interactuable applications, slightly decoupled from the context they are in. With increasing computing power, even in mobile devices, the big difference [50] in power needed to complete a task between JavaScript and other native languages starts getting negligible. GWT achieves **high compatibility** with all different browsers, and with the help of different APIs can be a very good tool for creating gadgets in situations where the concept of the gadget makes sense. The only limitation is it is not possible to get outside of the gadget apart from what the API provides.

Gadgets are a relatively **easy way to extend Wave**, as no actual knowledge of Wave's codebase is needed to do so. But still the Gadgets API is far from perfect, and requires the developer to repeat tasks for every gadget made. Also, there will be sometimes version compatibility problems, project configuration problems, classpath errors, and the messages given by the GWT compiler are not informative enough. Many steps need to be followed to get things working.

Also, even though the Gadgets API is a Java API, the fact that it generates JavaScript code means troubleshooting is sometimes needed in the JavaScript layer, so JavaScript knowledge is also needed, and computer generated code can sometimes not be easy to follow and understand.

The project aims to take advantage of the features in Wave and solve the issues and difficulties that could arise for developers when programming Gadgets and Robots. To achieve it, a framework for development and complete real extensions have been developed.

As a framework, successful development of both gadgets and robots has been carried out completely from start to finish, using **all the important and interesting features** provided by the Gadgets API and Robots API. Not only that, but also issues have been documented and samples have been provided to the public to use.

Useful gadgets have also been developed. They were inspired by **actual needs**, and have been finished to a ready to use state. Anyway, they are openly available under the Affero General Public License[4] for anyone to use, modify, and continue improving them.

10.3 Future Work

There is a lot of work that can be done to Wave, and the wave protocol has a big potential. Apache took control of the project, but the development from their part is slow and very conservative, so there is actually a **lack of work from part of a big player**. The most work is now being done by the P2Pvalue project and Kune, but when pushing changes to the Apache repository, the changes are not always accepted.

Wave is in need of features, and some of the ones done here can be considered essential, an they can always be implemented **closer to the Wave interface and more familiar** regarding the user experience:

- Robots: Maybe gadgets and robots are not user-friendly enough. If a user is interested in one of the features a robot can provide, he might not be interested in **adding an unknown user** to his wave which he wants to keep private.

- Gadgets: There is no way of **seeing all of the gadgets that are available**. If a gadget is not added to the list of gadgets, you need to know the exact URL of the gadget to use it, that means having to know the gadget beforehand. There are, too, many options when trying to choose a gadget, and you need to test them to see if they are the ones needed.

When developing the Decision Maker Gadget, because of its slightly complex state storing structure, it was clear that the way of storing state as key-value pairs is too limiting. Storing complex values as strings means you have to create a special format depending on the nature of the object to store, storing lists means having to use a character as a separator of the elements rendering that character unable to be used inside the elements themselves. A good solution for this would be to make a **steriliser library** that generates XML or JSON documents representing the structure and value of objects, and submits and retrieves values from the Wave's state in an easy way.

Bibliography

[1] Google Docs.
https://drive.google.com/.

[2] Zoho.
http://www.zoho.com/.

[3] Primavera De Filippi et al, Technical Report D1.3: Guidelines on design settings: Guidelines on institutional design settings, value metrics and value rewards systems, and CBPP software platform features. Public Deliverable, P2Pvalue Project (FP7-ICT-2013-10, 2013-2016, grant #610961), will be available in http://p2pvalue.eu).

[4] GNU. Affero General Public License.
http://www.gnu.org/licenses/agpl-3.0.html.

[5] Google. What happened to iGoogle?
https://support.google.com/websearch/answer/2664197.

[6] Gina Trapani and Adam Pash. *The complete guide to Google Wave.* 3ones Inc, 2010.

[7] Google. Wave Protocol.
http://www.waveprotocol.org/federation.

[8] Apache. Apache Wave (Incubating).
http://incubator.apache.org/wave/about.html.

[9] Google+.
https://plus.google.com/.

[10] Facebook. Facebook Games.
https://www.facebook.com/appcenter/category/games/.

[11] Adobe. Flash.
http://www.adobe.com/products/flash.html.

[12] Oracle. Applets.
http://www.oracle.com/technetwork/java/applets-137637.html.

[13] uNrEaL. uNrEaL BoT.
http://www.unrealbot.com/forum/.

[14] Suzanne Vranica. One-Third of Traffic Is Bogus.
The Wall Street Journal: online.wsj.com/news/articles/SB10001424052702304026304579453253860786362, 2014.

[15] Kune.
http://kune.ourproject.org/.

[16] Comunes.
http://comunes.org/.

[17] Google. Wave Federated Protocol.
http://www.waveprotocol.org/protocol/draft-protocol-specs.

[18] Peter Saint-Andre. Extensible messaging and presence protocol (xmpp): Core. 2011.
RFC 6120 Available at http://tools.ietf.org/html/rfc6120.

[19] Torben Weis and Arno Wacker. Federating Websites with the Google Wave Protocol. *Internet Computing, IEEE*, 15(3):51–58, 2011.

[20] Bruce W Perry. *Google web toolkit for ajax.* " O'Reilly Media, Inc.", 2007.

[21] Google. Gadgets API.
https://developers.google.com/gadgets/.

[22] Google. Robots API.
http://www.waveprotocol.org/wave-apis/google-wave-robots-api-overview.

[23] Eclipse.
http://www.eclipse.org/.

[24] OpenJDK.
http://openjdk.java.net/.

[25] Jetty.
http://www.eclipse.org/jetty/.

[26] Apache. Tomcat.
http://tomcat.apache.org/.

[27] Chromium.
http://www.chromium.org/.

[28] Iceweasel.
https://wiki.debian.org/Iceweasel.

[29] Apache. Maven.
http://maven.apache.org/.

[30] Juliana Mitchell-Wong, R Kowalczyk, A Roshelova, B Joy, and H Tsai. Opensocial: From social networks to social ecosystem. In *Digital EcoSystems and Technologies Conference, 2007. DEST'07. Inaugural IEEE-IES*, pages 361–366. IEEE, 2007.

[31] Google. Google Wave Federation Architecture White Paper.
http://wave-protocol.googlecode.com/hg/whitepapers/google-wave-architecture/google-wave-architecture.html.

[32] Google. Google Wave Federation Protocol Over XMPP.
http://wave-protocol.googlecode.com/hg/spec/federation/wavespec.html.

[33] Hans-Erik Eriksson and Magnus Penker. Business modeling with UML. *Business Patterns at Work. New York et al*, page 17ff, 2000.

[34] Martin Fowler. Inversion of control containers and the dependency injection pattern, 2004.
Available at https://blog.itu.dk/MMAD-F2013/files/2013/02/3-inversion-of-control-containers-and-the-dependency-injection-pattern.pdf.

[35] Google. GWT Internationalization.
http://www.gwtproject.org/doc/latest/tutorial/i18n.html.

[36] David A. Wheeler. Why Open Source Software / Free Software (OSS/FS, FLOSS, or FOSS)? Look at the Numbers!). http://www.dwheeler.com/oss_fs_why.html. Revised as of May 8, 2014.

[37] Creative Commons. *The Power of Open*. 2011. Also available at http://thepowerofopen.org/assets/pdfs/tpoo_eng.pdf.

[38] P2Pvalue Wiki. Gadget Development Tutorial. http://grasia.fdi.ucm.es/p2pvalue/redmine/projects/kune/wiki/Gadget_Development_Tutorial.

[39] Creative Commons License Chooser. https://creativecommons.org/choose/.

[40] Antonio Tenorio-Fornés and Samer Hassan. Towards an Agent-supported Online Assembly: Prototyping a Collaborative Decision-Making Tool. In *COLLA 2014, The Fourth International Conference on Advanced Collaborative Networks, Systems and Applications*, 2014. Accepted.

[41] appear.in. https://appear.in/.

[42] Alan B Johnston and Daniel C Burnett. *WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web*. Digital Codex LLC, 2012.

[43] P2Pvalue Wiki. Robot Development Tutorial. http://grasia.fdi.ucm.es/p2pvalue/redmine/projects/kune/wiki/Robot_Development_Tutorial.

[44] P2Pvalue Repository. Robot Basic Sample README. https://gitorious.org/p2pvalue/waverobotbasicsample/source/a99e76e234935a2290b881c5e50434e6260ff2d6:.

[45] P2Pvalue Redmime. Robot Registration Process. http://grasia.fdi.ucm.es/p2pvalue/redmine/projects/kune/wiki/Robot_Registration_Proccess.

[46] Creative Commons Gadget Repository.
https://github.com/End-of-degree-project/Gadget-Creative-Commons.

[47] Decision Maker Gadget Repository.
https://github.com/End-of-degree-project/Gadget-Decision-Maker.

[48] Video Conference Gadget Repository.
https://github.com/End-of-degree-project/Gadget-Video-Conference.

[49] Robot Colorizer Repository.
https://github.com/End-of-degree-project/Robot-Colorizer.

[50] Harry Fairhead. JavaScript Is Slow.
http://www.i-programmer.info/news/86-browsers/3492-javascript-is-slow.html, 2011.